# Automation of Software Testing for Reusable Components

Jongseok Kim[1], Kamel Rekab[2], Lotfi Tadj[*3]

Department of Computer Science, Department of Mathematics and Statiatics, School of Business Administration
Florida Institute of Technology, University of Missouri-Kansas City, Dalhousie University, USA

Dalhousie University, School of Business Administration, Halifax, NS, Canada

[1]kim_jongseok@yahoo.com; [2]rekabk@umkc.edu; [*3]lotftadj @yahoo.com

***Abstract***

Reusing software components without proper analysis is very risky because software components can be used differently from the way that they were developed. This paper describes a new testing approach for reusing software components. With this new approach, it is possible to automatically decide if software components can be reused without testing. In addition, when retesting is required for reusing software, test cases are generated more efficiently using the previous testing history.

***Keywords***

*Software Testing; Software Reuse; Markov Chain; Statistical Testing*

## Introduction

The reusability of software components is a major issue for software developers because reusing software can save development time and effort. In addition, it can reduce errors (Basili et al., 1996). Generally, reusable software components are regarded as safe because they are tested during development. However, reusing software components can be risky if they are not reanalyzed for reuse. Even though software components have passed testing in the original development environments, there can be problems in reusing the components in their new environment. Therefore, reanalysis of software components must be required before they are reused, and often, reanalysis of software components requires additional tests. For testing software for reuse, test case generation methods are needed. In addition, new measurements are needed to decide if the software components need more testing before reusing and when retesting should stop. This paper describes a new approach to solve problems in testing software for reuse. This new approach is based on studies of software testing using a Markov chain (Oshana, 1997; Whittaker, 1992; Whittaker and Thomason, 1994).

This paper consists of four sections. Section 1 is the introduction. In Section 2, the new approach for testing software for reuse is described, and in Section 3, the case study for the new approach is summarized. Section 4 provides the overall conclusion.

## Testing Procedure for Software Reuse

It is commonly known in software industry that there is no perfect software testing method. Therefore, software engineers are constantly trying to find an appropriate if not better software testing solution. Generally, reliable software testing is testing that finds many errors, but this concept is too abstract. We define reliable testing differently. Generally, software testing is defined as the process of demonstrating correspondence between a program and its specification (Beizer, 1996; Pressman, 1992). Therefore, the procedure for testing software can be considered as a simulation of the software usage, and when the software is tested as close as possible to the real usage of the software, we can then say that it is reliable software testing. Our approach for software testing is based on this idea. In this section, the procedure of the new approach is described.

Our testing approach to software reuse consists of five steps. In the first step, the usage model for the new environment is created when a software component is reused. The usage model is a Markov chain that shows the expected behavior of a software component in the new environment. In the second step, the initial test cases are generated and executed if the previous test history is not available. The third step is to create and update the test model, which keeps the previous test history. The fourth step is to measure the stopping criterion using the difference between the usage model and the test model. In this step, it will be decided whether or not retesting is required or not when a software component is reused. The fifth step it to

generate and to execute test cases for retesting a software component when additional tests are required. After generating and executing test cases for software reuse, the fourth step is repeated to decide if the software component needs more testing before reusing it. Figure 1 illustrates the general process of the approach.

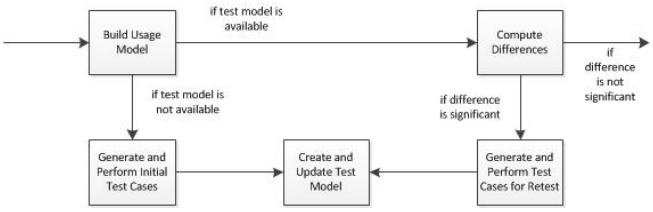



FIG. 1 OVERVIEW

### *Usage Model Creation*

The usage model (Becker and Whittaker, 1997; Oshana, 1997; Whittaker, 1992) is a Markov chain that simulates the actual usage of a software component by assigning probabilities on arcs based on the behavior of a software component. The usage model consists of states and arcs. The states represent statuses of the software behavior, and all states must be unique to present different statuses of the software behavior. In the usage model, the states must obey the Markov property that the next state is determined independently of the previous states; furthermore, all states are connected by the directed arcs. The arcs in the usage model usually represent inputs applied to a software component. Inputs can come from users, systems, and other software components; thus, they can cause transitions from one state to other states. Because there can be a number of inputs to a software component, the abstraction technique such as the category partition method (Ostrand and Balcer, 1988) can be used. Each arc in the usage model is also associated with a probability to simulate the actual usage. There are several ways to assign probabilities into arcs, such as real data from the prototypes, customer surveys, or the judgment of domain experts (Oshana, 1997). However, in the case where the actual usage is unknown, the probabilities can be assigned uniformly (Whittaker, 1992).

Using the usage model, it is possible to simulate the new environment for a software component by adjusting the probabilities on the arcs. Generally, when a software component is reused in a new environment, the functions of the software component are not changed. Only their usages are changed. Therefore, the usage model can be rebuilt by assigning different probabilities to reflect the new environment.

(Sriavastava, Jose, Barde and Gosh, 2010) also used a Markov chain based usage model that describes software on the basis of its statistical usage data. Their paper proposed a technique to generate optimized test sequences from a Makov chain based usage model.

Software reliability test based on Markov usage model was studied by (Zhou, Wang, Hound and Ai, 2012). Their paper focused on the generation of Markov usage model of software system and the method of software reliability test based on the Markov usage model.

(Bettinotti, Garavaglia 2011) described an application of statistical testing with Markov chain usage model to programmers and testers during web site development, to guarantee the software reliability. Their article was targeted at test automation for websites development with JUMBL and JWebunit.

### *Initial Test Case Generation*

This step is performed only when the previous test model is not available. To create the initial test model, several test case generation methods using the usage model can be used in this step, such as all path generation and statistical methods. In this paper, the random test case generation, which is one of the statistical methods, is used to generate the initial test cases for the case study.

The random test case generation method is a very popular back-box testing technique in statistical methods. In addition, because it generates test cases close to the actual usage of a software component using probabilities associated with arcs in the usage model, the test model becomes similar to the usage model after testing. Moreover, according to Duran and Ntafos' paper (Duran and Ntafos, 1981), it is more cost-effective for many programs including real-time software.

In the random test case generation, the next state and the next input in a test case are chosen randomly based on the probabilities associated with arcs in the usage model. Thus, if an arc has a high probability, it has more of a chance to be chosen. The probabilities might be assigned manually based on the available usage history or the user preference for a software component. In other cases, the probabilities can be assigned uniformly. In this method, test case generation is halted when all states and arcs are covered.

After the initial test is performed, all test results are recorded on the test model (Becker and Whittaker, 1997; Whittaker, 1992), which has the same structure of the usage model but has frequencies instead of probabilities.

***Test Model Update***

The test model (Becker and Whittaker, 1997; Whittaker, 1992) is a Markov chain that maintains the previous test information. This model is critical in finding the areas where more tests are needed when a software component Is reused. The test model is created by initializing the usage model with the same states and arcs; the difference being that it uses the frequency counts instead of the probabilities in the usage model as the arc labels. Initially, the frequencies of the arcs in the test model are set to zero, and whenever inputs are used in test cases during testing, the frequency of the corresponding arc increases by one.

Moreover, to handle errors, the failure state is used, so whenever an error occurs, the frequency of the arc between the current state and the failure state increases by one. If an error is critical, running the test case is aborted, and the arc from the failure state goes to the termination state. If an error is not critical, the test sequence continues, and the arc from the failure state goes to the next state of the test sequence. This scheme helps to increase the difference between the usage and the test models whenever errors occur. This test model is updated whenever tests are performed on a software component.

In this study, the original test model (Becker and Whittaker, 1997; Whittaker, 1992) is modified. In this new approach, multiple test models are maintained instead of a single test model.

***1) Multiple Test Models***

The problem of using a single test model is that unnecessary retesting may be performed when a software component is reused. For example, sometimes, the difference between the new usage model and the test model may be significant although a very similar usage was previously used. It is possible that a software component has been reused in very different environments since a similar usage model was used, so the test model has been updated differently. Because of this, the difference between the new usage model and the test model is high even though a similar one was used before. In this case, unnecessary testing is required to reduce the difference between the new usage model and the test model.

To avoid this problem, all test models must be kept separately instead of using a single test model to keep the test history. Therefore, when a software component is reused, the new usage model is compared to the history of the test models. If there is a similar test model to the new usage model, a software component is reused without retesting. However, if there is no test model that is similar enough, retesting is necessary.

In this approach, all test models are kept in addition to the main test model that is the same as the test model in another article (Becker and Whittaker, 1997; Whittaker, 1992), and test cases are generated based on the most similar test model. When test cases are applied, not only the most similar test model is updated but also the main test model is updated, and both updated test models are kept in the test history. Using this approach, it is possible to find out if there is a similar usage model that was used before. When test case generation is needed for the new usage environment, the number of test cases can be reduced because test cases are generated based on the most similar test model.

For example, when a software component is reused for the first time, there is only a single test model, the main test model *MT*. Therefore, the new usage model is compared to *MT*. If they are similar enough, a software component is reused without retesting. Otherwise, retesting is performed before a software component is reused. In this case, test cases are generated based on *MT* because there is only a single test model, and the new test model $TM_1$ is created based on *MT*. When $TM_1$ is created, all frequencies in *MT* are used as the base frequencies in $TM_1$, and both MT and $TM_1$ are updated with retesting results. When a software component is reused for the second time, the new usage model is compared to both *MT* and $TM_1$. At this time, the two test models are actually the same. If the difference is high, the test cases are generated based on either MT or $TM_1$, and a new test model is $TM_2$ created. In addition, *MT* and $TM_2$ are updated using retesting results. During the third time of reusing a software component, the new usage model is compared to all three test models, MT, $TM_1$ and $TM_2$, and if there is one that is similar enough to a new usage model, no retesting is performed. Otherwise, test cases are generated

based on the closest test model, and a new test model $TM_3$ is created using the test model that is the most similar one to the new usage model. In addition, *MT* and $TM_3$ are updated. The main test model *MT* keeps the whole history of testing to compute the reliability, so the main test model *MT* is updated every time test cases are applied. Figure 2 illustrates the updating of the test model.

Using this method, all test models can be maintained in addition to the main test model, so when a software component is reused, it is possible to find out if a very similar usage model has been previously used. In the case where retesting is required, the number of test cases can also be reduced because the test cases are generated based on the closest test model.

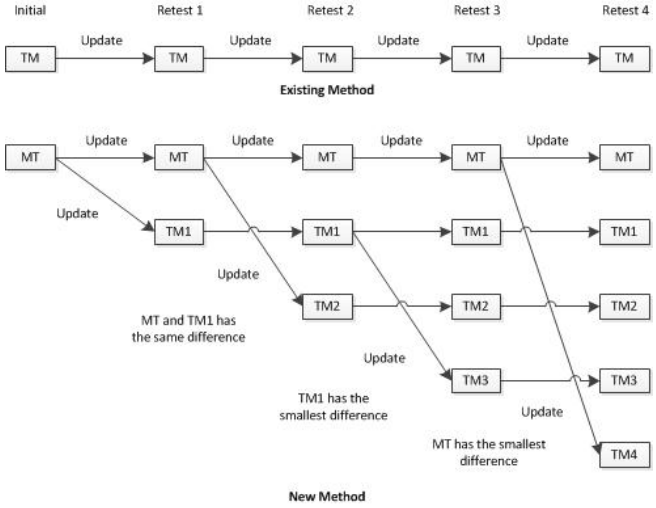



FIG. 2 UPDATING THE TEST MODEL

### *Stopping Criterion*

When a software component is reused, the same stopping criterion that is used in the software development process is not very useful because a software component must satisfy the stopping criterion before its release. Therefore, a new stopping criterion is needed for software reuse.

In this paper, the difference between the usage model and the test model Is used as the stopping criterion for software reuse. When a software component is reused, the new usage model might have different arc probabilities from the previous usage model. Therefore, in order to find that the new usage model is similar enough to the test model, the difference is measured. If the usage and test models are similar, it means that a software component has been tested according to the actual usage in the new environment. In this case, no more tests are required to reuse a software component. However, if the difference is significant, additional tests are required because a software component has been tested significantly different from the usage in the new environment. In this case, retesting is performed until the test model is similar enough to the new usage model. Therefore, the simple linear model is used to measure the difference between the usage model and the test model focusing on the inputs.

#### *1) Simple Linear Regression*

Regression analysis is generally used to describe the relationship between two or more variables (Harnett, 1982), and linear regression is used to show variables are linearly related. In this study, simple linear regression is used to show whether or not the usage model and the test model are the same. In this approach, the usage model becomes the independent variable *X*, and the test model becomes the dependent variable *Y*. In addition, the expected frequencies are computed for the arcs in the usage model. To show that the difference between two models is not significant, the intercept ($\beta_0$) and the slope ($\beta_1$) are estimated using the least-squares method, and we check if the intercept is close to zero and if the slope is close to one. In addition, to check how precise the estimated slope and intercept are, the confidence intervals for the slope and the intercept must be estimated.

Since both the slope and the intercept are considered in our problem, the joint confidence intervals for the slope and the intercept should be calculated. In this study, the Bonferroni method (Neter et al., 1985) is used. The Bonferroni method is very simple and guarantees joint confidence intervals of at least $\alpha$ when the slope and the intercept are separately estimated with confidence intervals for $\alpha/2$. The confidence intervals are useful in checking the precise estimated values. If the confidence intervals are small, the estimated value is more precise.

To decide the difference between the usage model and the test model, the hypothesis test is performed. In this study, we want to check that the usage and the test models are the same. To be the same, the slope must be one, and the intercept must be zero. Therefore, the null hypothesis and the alternative hypothesis are as follows:

$$H_0: \quad \beta_0 = 0 \quad \text{and} \quad \beta_1 = 1,$$
$$H_\alpha: \quad \beta_0 \neq 0 \quad \text{or} \quad \beta_1 \neq 1.$$

The values $t_0$ of $t_1$ and for the hypothesis test are computed as follows:

$$t = \frac{a}{S_a}, \quad t = \frac{b-1}{S_b},$$

where $a$ is the least square estimator of $\beta_0$, $b$ is the least square estimator of $\beta_1$, $S_a$ is the standard error of $a$, and $S_b$ is the standard error of $b$. If both $t$ values are between $-t(1-\frac{\alpha}{4};n-2)$ and $t(1-\frac{\alpha}{4};n-2)$, the null hypothesis is accepted with (1 - $\alpha$) confidence, and it is concluded that the usage model and the test model are the same. This being the case, a software component can be reused without additional testing. Otherwise, the null hypothesis is rejected with (1 - $\alpha$) confidence, and it is concluded that the usage model and the test model are different. In that case, retesting is required before a software component can be reused. Once it is retested, the hypothesis testing is performed to check if the usage model and the test model are similar.

***Test Case Generation of Retest***

When a software component needs additional testing for software reuse, it is possible to use existing test case generation methods. However, the problem with using the existing methods is that unnecessary testing may be performed because they do not use the previous testing information when generating test cases. If it is possible to use the previous testing information to generate test cases, then unnecessary testing can be avoided.

The purpose of this step is to minimize the differences between the usage model and the test model using a smaller number of new test cases. In this step, the random test case generation method is also used like in initial test case generation. However, the difference of each arc between the usage and the test models is used to generate test cases instead of the probability in the usage model. When the difference is used for testing, it is useful to find which areas need more testing, and more test cases are generated from the areas with high differences.

Generally, the differences $d$ between the usage model $UM$ and the test model $TM$ are calculated by subtracting the frequencies of the arcs in the test model from the expected frequencies of the arcs in the usage model.

$$d_{i,j} = UM_{i,j} - TM_{i,j}.$$

In this case, the differences can be categorized into three groups of values, positive, negative, or zero, and they can have different meanings. The positive values mean that more tests are required in the arcs. The negative values show that too many tests have been performed on the arcs. The zero value means that the proper amount of tests has been performed on the arcs. Therefore, only positive values are needed to generate test cases because the areas with positive values need additional testing. However, the information about the arcs with negative values should be considered for the test case generation. To keep the order of the differences and to make all values of the differences positive, all differences are subtracted by the smallest value. In this case, the smallest difference becomes zero, but this can cause a problem when generating test cases. When the value is zero, the arc with a zero difference is not used to generate test cases. Therefore, if the arc with a zero difference is only the arc from one state to another state, generating test cases might be problematic because some test case sequences cannot be completed. Therefore, to avoid this problem, we must add one to all the differences to eliminate the zero value. Therefore, the differences used in generating test cases are computed as follows:

$$d'_{i,j} = d_{i,j} - \min_j \left(d_{i,j}\right) + 1.$$

After the differences for all arcs are calculated, these differences can be easily converted and then used to generate test cases in the same manner as the random test case generation with probabilities. After the test case generation, testing on a software component is performed, and the test model is updated using the testing results.

## Case Study

For the case study, one of the Ada 95 Booch components (http://sourceforge.net/projects/booch95/) was used. Originally, Booch components were created for C++ and were ported to Ada by David Weller and members of Team ADA. Ada 95 Booch components implement data structures and are widely used in Ada programming. Booch components have 30 components including Bags, Collections, Graphs, Lists, Maps, Queues, Rings, Sets, Stacks, and Trees. We selected a double list in Booch components for the case study. It has 28 public, 4 generic, and 12 private functions and procedures.

In the case study, first, an initial usage model had to be created. Once the initial usage model was created,

then the initial test cases were generated. After that, the initial test was performed, and the initial test model was created using the test results. Once the initial usage and the initial test models were created, then the software component was tested for reuse.

***Usage Model Creation***

There are several ways to crate usage model, but in this case study, the usage model was created using the operational mode (Whittaker, 1997), which is a formal characterization of the status of one or more internal data objects that affect software behavior.

In the double list Booch component, there are two operational modes, number_of_lists and null_list, and they have values as follows:

no_of_lists = (0, 1, 2)

null_list = (null_both, null_1, null_2, not_null)

Using the operational modes, states in the usage model are created by the combination of the operational mode values. In this case, it is possible to create 12 states. However, there are some useless states when the number of lists is zero and 1. After the elimination of the useless states, the number of states is reduced to seven. In addition to these states, there are three system states, start state for the beginning of the component execution, error state for the error handling, and termination state for the normal termination of the component. Therefore, there are the following 10 states in the usage model.

1. (start) 2. (0)
3. (1, null_1) 4. (1, not_null)
5.(2, null_both) 6. (2, null_1)
7. (2, null_2) 8. (2, not_null)
9. (Error) 10. (Termination)

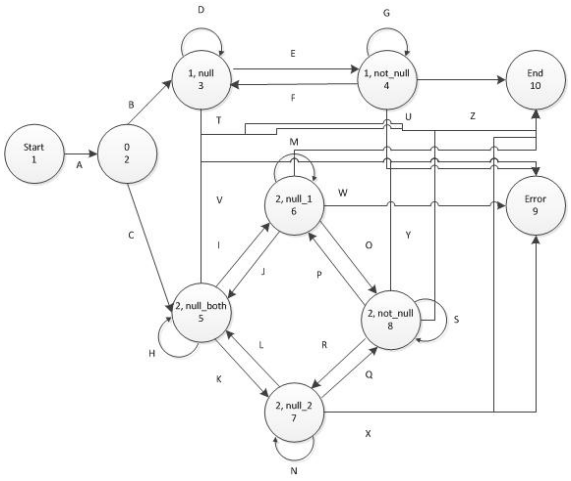



FIG. 3 USAGE MODEL FOR DOUBLE LIST

After the state creation, inputs should be defined. To define the inputs, the category partition method (Ostrand and Balcer, 1988) was used. Using the category partition method, 65 valid and 66 invalid inputs were defined for the double list Booch component. The following figure shows the usage model for the software component along with the inputs used in the usage model and their probabilities. In this model, uniform probabilities are used. Table 1 shows the inputs in the usage model.

TABLE 1 INPUTS

| API Group | No. of APIs | From State | To State | Probability for Each Input |
|---|---|---|---|---|
| A | 1 | 1 | 2 | 1 |
| B | 1 | 2 | 3 | 0.5 |
| C | 1 | 2 | 5 | 0.5 |
| D | 4 | 3 | 3 | 0.045455 |
| E | 4 | 3 | 4 | 0.045455 |
| F | 4 | 4 | 3 | 0.026316 |
| G | 21 | 4 | 4 | 0.026316 |
| H | 4 | 5 | 5 | 0.027027 |
| I | 5 | 5 | 6 | 0.027027 |
| J | 5 | 6 | 5 | 0.014286 |
| K | 5 | 5 | 7 | 0.027027 |
| L | 5 | 7 | 5 | 0.014286 |
| M | 24 | 6 | 6 | 0.014286 |
| N | 24 | 7 | 7 | 0.014286 |
| O | 8 | 6 | 8 | 0.014286 |
| P | 6 | 8 | 6 | 0.016949 |
| Q | 8 | 7 | 8 | 0.014286 |
| R | 6 | 8 | 7 | 0.016949 |
| S | 32 | 8 | 8 | 0.016949 |
| T | 13 | 3 | 9 | 0.045455 |
| U | 12 | 4 | 9 | 0.026316 |
| V | 22 | 5 | 9 | 0.027027 |
| W | 32 | 6 | 9 | 0.014286 |
| X | 32 | 7 | 9 | 0.014286 |
| Y | 14 | 8 | 9 | 0.016949 |
| Z | 1 | 3 | 10 | 0.045455 |
| | | 4 | 10 | 0.026316 |
| | | 5 | 10 | 0.027027 |
| | | 6 | 10 | 0.014286 |
| | | 7 | 10 | 0.014286 |
| | | 8 | 10 | 0.016949 |

*Initial Test Case Generation*

Because the testing history and the test model were not available for the Booch components, the initial test cases were generated, and the initial test model was created using the test results. In the initial test case generation, any test case any test case generation methods using the usage model can be applied, such as random test case generation and all path generation methods. The random test case generation method is a very popular method in statistics, and all path generation gives the maximum coverage of the model.

In this case study, the random test case generation method was used to generate the initial test cases. In this particular case, it generated 2588 test cases. Using these results, the initial test model is created.

*Results Analysis*

The first step was to check to see if the simple linear regression method was appropriate for our problem, so the aptness of model test was performed. Second, to check the efficiency of the new method, the results of the new test case generation method were compared to the results of the random test case generation.

*1) Aptnres of Model*

To check if the simple linear regression method was appropriate for our problem, the following four types of departures were considered using the residual analysis (Neter et al., 1985; Rekab, 1998).

1. The regression function is not linear.
2. The error terms do not have constant variance.
3. The error terms are not independent.
4. The error terms are not randomly distributed.

In addition to the residual analysis, an ANOVA (Analysis of Variance) table was created and used to test if there was a linear relationship. The results were analyzed after 10,000 test cases were run.

Using simple graphic analysis of residuals, the first three types of departure can be verified (Bowerman et al., 1986; Neter et al., 1985; Rekab, 1998). The following figures show the residual against the fitted test model for the new method and the existing method.

In Figures 4 and 5, it shows that the residuals have the tendency to fall within a horizontal band around zero, so the regression function is linear. Moreover, it displays that the predicted values are close to the observed values, so the error terms have a constant variance. In addition, because the residuals seem to randomly fluctuate around zero, the error terms are independent. Therefore, the residual plot shows that the first three departures do not occur in our problem.

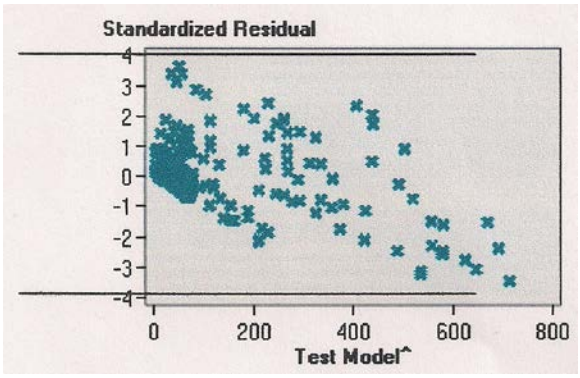



FIG. 4 RESIDUAL PLOT FOR NEW METHOD

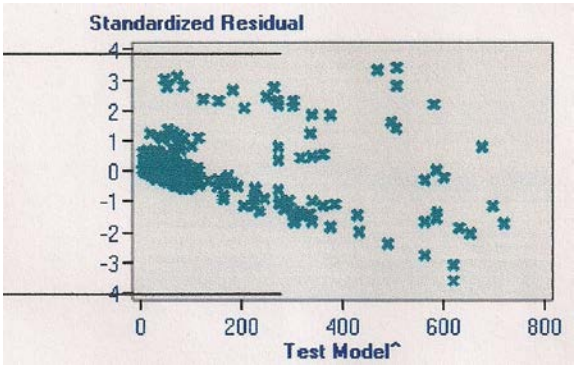



FIG. 5 RESIDUAL PLOT FOR EXISTING METHOD

To check the normality of the error terms, histograms of the residuals for the new method and the existing method are created as follows:

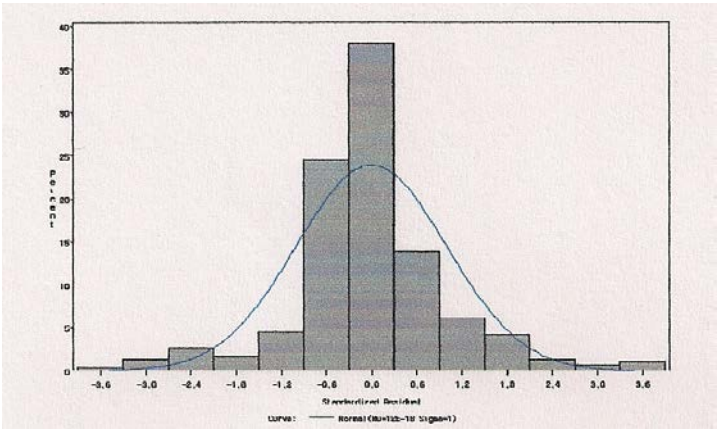


FIG. 6 HISTOGRAM OF RESIDUALS FOR NEW METHOD

Figures 6 and 7 clearly show that the error terms are normally distributed in both methods.

Generally, ANOVA (Analysis of Variance) table

can provide an overall summary of regression analysis. ANOVA table consists of several estimates of variance, and they can be used to answer the principal questions of regression analysis (Kleinbaum and Kupper, 1978). For simple linear regression, the estimate of variance in ANOVA table is useful to find if there is a straight-line relationship.

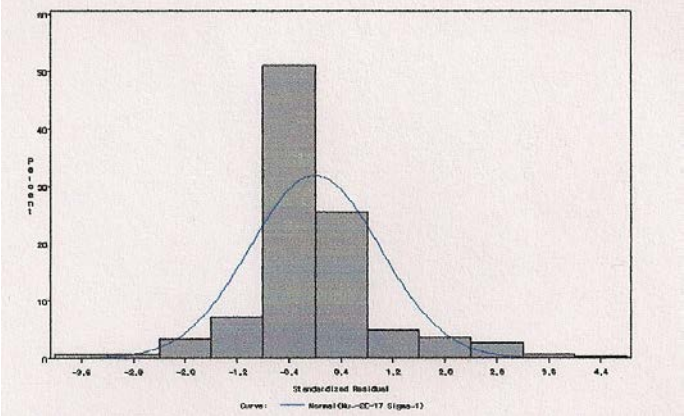


FIG. 7 HISTOGRAM OF RESIDUALS FOR EXISTING METHOD

After 10,000 test case runs, the ANOVA tables for the new method and the existing method were created as follows:

TABLE 2 ANOVA TABLE FOR NEW METHOD

| Source of Variation | SS | Df | MS | F |
|---|---|---|---|---|
| **Regression** | 242730806.17 | 1 | 242730806.17 | 37784.58 |
| **Error** | 1978613.581 | 308 | 6424.070 | |
| **Total** | 244709419.76 | 309 | | |

TABLE 3 ANOVA TABLE FOR EXISTING METHOD

| Source of Variation | SS | Df | MS | F |
|---|---|---|---|---|
| **Regression** | 240563387.44 | 1 | 240563387.44 | 26575.61 |
| **Error** | 2488027.21 | 308 | 9052.04 | |
| **Total** | 243351414.65 | 309 | | |

Using the computed values of $F$, the following hypothesis test is performed for the new method and the existing method.

$$H_0: \quad \beta_1 = 0,$$
$$H_\alpha: \quad \beta_1 \neq 0.$$

In this hypothesis test, if the computed $F$ is smaller than $F(1 - \alpha; 1, n - 2)$, $H_0$ is accepted, and it is concluded that there is no significant linear relationship with $(1 - \alpha)$ confidence. However, if the computer $F$ is greater than $F(1 - \alpha; 1, n - 2)$, $H_0$ is rejected, and it is concluded that there is a linear relationship. In this particular case, the value of $F(0.95; 1, 308)$ is 3.84, and it is smaller than the value of $F$. Therefore, the hypothesis is rejected with 0.95 confidence, so it can be concluded that there is a linear relationship for both methods. However, the value of $F$ in the new method is greater than one in the existing method, so it is possible to conclude that when the new method is used, the linear relationship is more significant.

According to the results of the residual analysis and the ANOVA table, it is clearly shown that the four types of departures do not exist in our problem and that there is a linear relationship in the new method and the existing method. Therefore, it can be concluded that the simple linear regression method is appropriate for our problem.

### 2) *Reducing Differences*

In the new approach, it is expected that the difference between the usage model and the test model is more efficiently reduced. Therefore, in this case study, the results of the new test case generation method are compared to the results of the random test case generation method to show how efficiently the new approach is in reducing the difference. In this experiment, only one usage model is used, and although the difference has reached the acceptance points, we continuously run a certain number of retests to monitor how the differences are reduced. To compare the results, the intercept, the confidence intervals for the intercept, the slope, and the confidence intervals for the slope are monitored at every 2,000 test cases. These plots help to see how both the usage model and the test model become close in both methods. In this study, 90% confidence interval for the slope and the intercept is used.

Numerical results for the difference between the usage model and the test models are presented in Tables 4 and 5. Table 4 presents the slopes, the 90% confidence intervals and the width of the confidence intervals when the new method is used. Whereas Table 5 presents the slopes, the 90% confidence intervals and the width of the confidence intervals when the existing method is used. Also a graphical representation of Tables 4 and 5 is presented in Figures 8, 9, 10, and 11.

Figures 8, 9, 10, and 11 display the range of the slope, the intercept, and their widths of the confidence intervals for the new test case generation and the random test case generation.

Figure 12 illustrates the initial scatter plot for the test model and the fitted model for the new test case generation method. The scatter plot of the usage model vs. the test model remains the same regardless of the number of test cases.

Scatter plots of the usage model vs. the new test model were performed after 6,000 test cases, after 12,000 test cases, after 18,000 test cases, after 24,000 test cases, and after 30,000 test cases. See (Kim, 2003).

It is shown in (Kim, 2003) that as the number of test cases increases the usage model and the new test model are nearly equal, the slope is close to 1 and the intercept is close to 0. In this article, we shall present the scatter plot of the usage model vs. the new test model after 30,000 test cases.

Similarly in (Kim, 2003), the usage scatter plots of the usage model vs. the existing test model were performed after 6,000 test cases, after 12,000 test cases, after 18,000 test cases, 24,000 test cases, and after 30,000 test cases.

TABLE 4 DIFFERENCE BETWEEN TEST CASES OF USAGE MODEL AND NEW TEST MODEL

| No. of Test Cases | Slope | Confidence Intervals | | | Intercept | Confidence Intervals | | |
|---|---|---|---|---|---|---|---|---|
| | | Low | High | Width | | Low | High | width |
| Initial | 0.93314 | 0.88572 | 0.98057 | 0.09485 | 5.17460 | -4.50623 | 14.85540 | 19.36123 |
| 2000 | 0.98302 | 0.95765 | 1.00839 | 0.05074 | 1.68873 | -6.97656 | 10.35400 | 17.33056 |
| 4000 | 0.98878 | 0.97158 | 1.00598 | 0.03440 | 1.36145 | -6.93931 | 9.66221 | 16.60152 |
| 6000 | 0.99152 | 0.97868 | 1.00437 | 0.02569 | 1.21444 | -6.80879 | 9.23767 | 16.04646 |
| 8000 | 0.99311 | 0.98289 | 1.00333 | 0.02044 | 1.14281 | -6.69723 | 8.98285 | 15.68008 |
| 10000 | 0.99418 | 0.98576 | 1.00259 | 0.01683 | 1.09675 | -6.55749 | 8.75099 | 15.30848 |
| 12000 | 0.99495 | 0.98781 | 1.00208 | 0.01427 | 1.06508 | -6.44548 | 8.57563 | 15.02111 |
| 14000 | 0.99555 | 0.98938 | 1.00172 | 0.01234 | 1.03873 | -6.33643 | 8.41389 | 14.75032 |
| 16000 | 0.99613 | 0.99075 | 1.00150 | 0.01075 | 0.99310 | -6.20173 | 8.18792 | 14.38965 |
| 18000 | 0.99653 | 0.99172 | 1.00133 | 0.00961 | 0.97254 | -6.14714 | 8.09222 | 14.23936 |
| 20000 | 0.99687 | 0.99256 | 1.00119 | 0.00863 | 0.94848 | -6.06203 | 7.95899 | 14.02102 |
| 22000 | 0.99717 | 0.99325 | 1.00109 | 0.00784 | 0.92710 | -6.00271 | 7.85691 | 13.85962 |
| 24000 | 0.99740 | 0.99379 | 1.00100 | 0.00721 | 0.91551 | -5.97525 | 7.80627 | 13.78152 |
| 26000 | 0.99759 | 0.99427 | 1.00091 | 0.00664 | 0.90672 | -5.91130 | 7.72474 | 13.63604 |
| 28000 | 0.99781 | 0.99475 | 1.00086 | 0.00611 | 0.87905 | -5.83789 | 7.59598 | 13.43387 |
| 30000 | 0.99798 | 0.99513 | 1.00082 | 0.00569 | 0.86188 | -5.79628 | 7.52003 | 13.31631 |

TABLE 5 DIFFERENCES BETWEEN TEST CASES OF USAGE MODEL AND EXISTING TEST MODEL

| No. of Test Cases | Slope | Confidence Intervals | | | Intercept | Confidence Intervals | | |
|---|---|---|---|---|---|---|---|---|
| | | Low | High | Width | | Low | High | Width |
| Initial | 0.93314 | 0.88572 | 0.98057 | 0.09485 | 5.17460 | -4.50623 | 14.85540 | 19.36123 |
| 2000 | 0.96587 | 0.93837 | 0.99337 | 0.05500 | 3.51306 | -5.92465 | 12.95080 | 18.87545 |
| 4000 | 0.97713 | 0.95775 | 0.99652 | 0.03877 | 2.95072 | -6.45433 | 12.35580 | 18.81013 |
| 6000 | 0.98371 | 0.96895 | 0.99846 | 0.02951 | 2.52148 | -6.74121 | 11.78420 | 18.52541 |
| 8000 | 0.98732 | 0.97542 | 0.99922 | 0.02380 | 2.28319 | -6.88404 | 11.45040 | 18.33444 |
| 10000 | 0.98962 | 0.97963 | 0.99961 | 0.01998 | 2.13505 | -6.98829 | 11.25840 | 18.24669 |
| 12000 | 0.99165 | 0.98311 | 1.00018 | 0.01707 | 1.93239 | -7.08554 | 10.95030 | 18.03584 |
| 14000 | 0.99281 | 0.98530 | 1.00032 | 0.01502 | 1.84799 | -7.16741 | 10.86340 | 18.03081 |
| 16000 | 0.99361 | 0.98689 | 1.00033 | 0.01344 | 1.80486 | -7.22705 | 10.83680 | 18.06385 |
| 18000 | 0.99431 | 0.98826 | 1.00037 | 0.01211 | 1.74960 | -7.24838 | 10.74760 | 17.99598 |
| 20000 | 0.99492 | 0.98943 | 1.00041 | 0.01098 | 1.69096 | -7.26114 | 10.64310 | 17.90424 |
| 22000 | 0.99530 | 0.99023 | 1.00038 | 0.01015 | 1.68699 | -7.31698 | 10.69100 | 18.00798 |
| 24000 | 0.99575 | 0.99106 | 1.00044 | 0.00938 | 1.63406 | -7.35709 | 10.62520 | 17.98229 |
| 26000 | 0.99580 | 0.99141 | 1.00019 | 0.00878 | 1.72487 | -7.32134 | 10.77110 | 18.09244 |
| 28000 | 0.99581 | 0.99165 | 0.99996 | 0.00831 | 1.82978 | -7.33542 | 10.99500 | 18.33042 |
| 30000 | 0.99599 | 0.99206 | 0.99992 | 0.00786 | 1.85523 | -7.37874 | 11.08920 | 18.46794 |

It is also shown in (Kim 2003) that as the number of test cases increases the usage model and the existing test model are also nearly equal. We shall present in our article only the scatter plot of the usage model vs. the existing test model after 30,000 test cases. However in each of the scatter plots investigated in (Kim 2003), the new test model is closer to the usage than the existing test model.

Figures 13 and 14 show the scatter plots for the test model and the fitted test model for the new test case generation method and the random test case generation after 30,000 test cases.

As shown in Figure 8, the values of the slope are getting close to one in both methods. However, the slope of the new method is always closer to one, and the new method is faster in becoming close to one. In the new method, the slope reaches 0.99 after 6,000 test cases, but when random test case generation is used, the slope reaches 0.99 after 12,000 test cases. In addition, when the new method is used, the width of the confidence intervals is always smaller as displayed in Figure 9. It shows that when the new method, the estimated slope is more precise.

Figures 10 and 11 illustrate the intercepts and their widths of the confidence intervals for both methods. They clearly show that when the new test case generation method is used, the intercept is always closer to zero and the widths of confidence intervals are smaller. In addition, in the new method, the intercept is continuously reducing to zero. In the random test case generation, the intercept is reducing to zero, but it is slower than the new method.

Figure 12 shows the initial scatter plot for the usage model and the test model. Figure 13 presents the scatter plot for the usage test model vs. the new test model after 30,000 test cases. It is clear that there is a linear relationship between the usage test model and the new test model. That is, the intercept is nearly zero and the slope is nearly equal to 1. A formal statistical test of significance needs to be performed, though.

Similarly Figure 14 shows the initial scatter plot for the usage model and the existing test model after 30,000 test cases. It is clear that there is a linear relationship between the usage test model and the new test model. That is the intercept is nearly zero and the slope is nearly equal to 1. A formal statistical test of significance needs to be performed, though.

However as we anticipated, the new test model is

closer to the usage model that the existing test model.

In our study, the Bonferroni method is used to compute the joint confidence intervals for stopping criterion. In our problem, we want to examine if the usage model and the test model are the same. In that case, the slope must be one, and the intercept must be zero when simple linear regression is used.

Therefore, to stop testing, the following hypothesis testing is performed:

$$H_0: \quad \beta_0 = 0 \quad \text{and} \quad \beta_1 = 1,$$
$$H_\alpha: \quad \beta_0 \neq 0 \quad \text{or} \quad \beta_1 \neq 1.$$

When the new test case generation method is used, the null hypothesis is accepted after 2,000 test cases 90% for the slope and the intercept. However, when the random test case generation is used, the null hypothesis is accepted after 12,000 test cases.

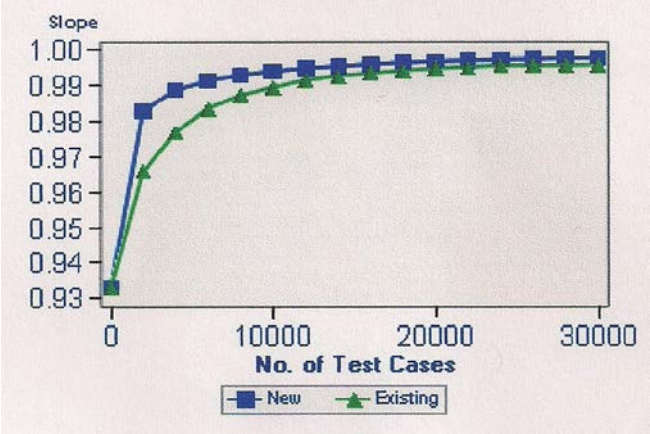



FIG. 8 SLOPE

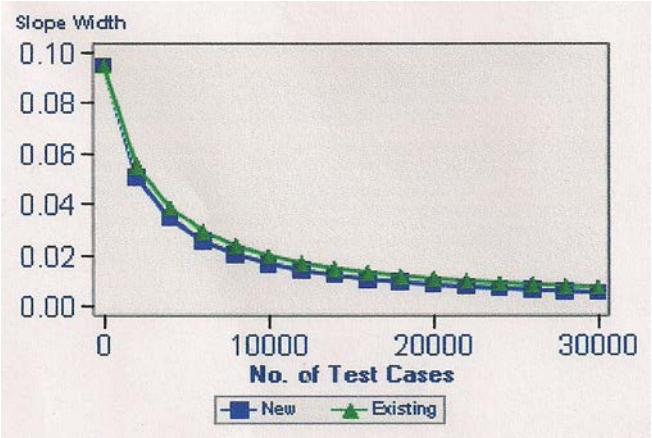



FIG. 9 WIDTH OF CONFIDENCE INTERVAL FOR SLOPE

According to the results, it is possible to conclude that the new method is more efficient in reducing the difference between the usage model and the test model. Therefore, with smaller test cases, the test model becomes similar to the usage model when the new test case generation is used.

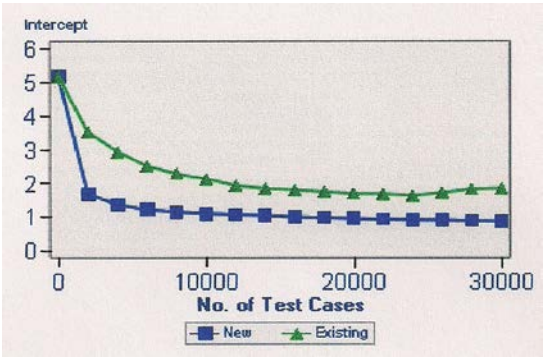



FIG. 10 INTERCEPT

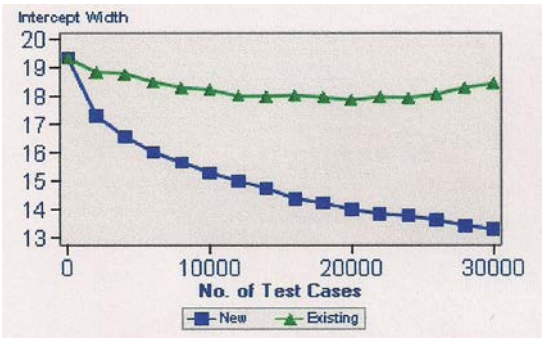



FIG. 11 WIDTH OF CONFIDENCE INTERVAL FOR INTERCEPT

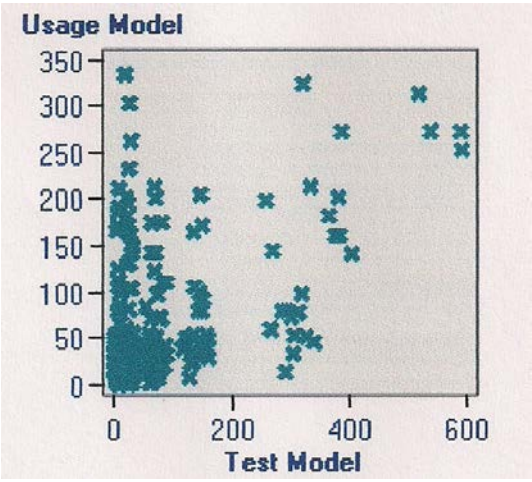



FIG. 12 INITIAL

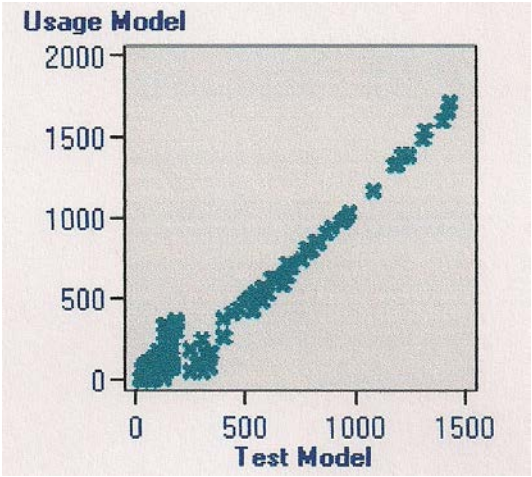



FIG. 13 SCATTER PLOT FOR NEW METHOD

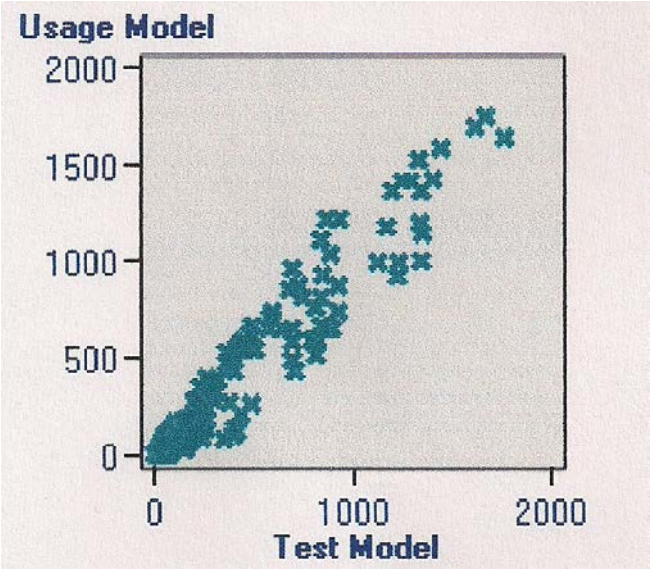



FIG. 14 SCATTER PLOT FOR EXISTING METHOD

## Conclusions

In this paper, the new approach to reusing a software component is developed to solve the problems in testing for software reuse. The new approach is based on studies of software testing using Markov chain (Oshana, 1997; Whittaker, 1992; Whittaker and Thomason, 1994), with several new methods added. First of all, the differences are used to generate test cases and to measure the stooping criterion of testing instead of the probabilities. In addition, multiple test models are used to keep the test history, and the average frequencies are used to update the test model.

The case study was performed using one of Booch Ada components. The case study shows that the new approach needs fewer test cases to reach the acceptance point when comparing to the existing method. For both methods, the slope gets close to one, and the intercept is getting close to zero when test cases are applied. However, when the new method is used, the results are always better, and the estimated values are more precise.

This new approach will be useful for software engineers when they reuse software components. In this method, the statistical approach is used to decide if more testing is needed for software reuse based on the information of the usage model and the test model. In addition, when additional testing is required, the differences are used to generate test cases. Because all methods use a mathematical approach, it is easy to automate. The automation will help engineers save time and effort in reusing software components.

## ACKNOWLEDGMENT

We are very thankful to the referee for very helpful suggestions that led to a significant improvement of our article.

## REFERENCES

Basili, V.R., Briand, L.C., and Melo, W. "How reuse influences productivity in object-oriented system." Communication of the ACM 39 (1996): 104-116.

Becker, S.A. and Whittaker, J.A., Cleanroom Software Engineering Practices. Harrisburg: IDEA Publishing, 1997.

Beizer, B. Software System Testing and Quality Assurance. New York: International Thompson Publishing Inc., 1996.

Bettinoti, M. and Garavaglia, M. "Test automation for Markov chain usage models." Journal of Computer Science & Technology 11 (2011): 93-98.

Bowerman, B.L., O'Connell, R.T., and Dickey, D.A., Linear Statistical Models: An Applied Approach. Boston: Duxbury Press, 1986.

Duran, J.W. and Ntafos, S., A report on random testing." Proceedings of the 5th International Conference on Software Engineering, San Diego, CA, March 9-12, 179-183, 1981.

Harnett, D.L. Statistical Methods, 3rd Edition. Reading: Addison-Wesley Publishing Company, 1982.

Kim, J., "Automatic Testing for Software Reuse." PhD Diss., Florida Institute of Technology, 2003.

Kleinbaum, D.G. and Kupper, L.L. Applied Regression Analysis and Other Multivariable Methods. Boston: Duxbury Press, 1978.

Neter, J., Wasserman, W., and Kunter, M.H., Applied Linear Statistical Models, 2nd Edition. Homewood: Richard D. Irwin Inc., 1985.

Oshana, R. "Software testing with statistical usage based models." Embedded Systems Programming January (1997): 40-55.

Ostrand, T.J. and Balcer, M.J. "The category-partition method for specifying and generating functional tests." Communication of ACM 31 (1988): 676-686.

Pressman, R., Software Engineering: A Practitioner's Approach, 3rd Edition. New York: McGraw-Hill, 1992.

Rekab, K., "Design of Experiments Made Easy." Lecture Notes, Florida Institute of Technology, 1998.

Srivastava, P.R., Jose, N., Barade, S., and Gosh, D. "Optimized test sequence generation from usage models using ant colony optimization." International Journal of Software Engineering & Applications 1 (2010): 14-28.

Whittaker, J.A. "Markov Chain Techniques for Software Testing and Reliability Analysis." PhD Diss., University of Tennessee, 1992.

Whittaker, J.A., "Stochastic software testing." Annals of Software Engineering 4 (1997): 115-131.

Whittaker, J.A. and Thomason, M.G. "A Markov model for statistical software testing." IEEE Transaction on Software Engineering 20 (1994): 812-824.

Zhou, K., Wang, X., Hou, G., Wang, J., and Ai, S. "Software reliability based on Markov usage model." Journal of Software 7 (2012): 2061-2068.

http://sourceforge.net/projects/booch95/